Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: 1, Solution Design in C++
Due date: March 30, 2021, 16:00
Lecturer: Dr. Markus Lumpe

Your name: Khang Trinh **Your student ID:** 102118468

Check Tutorial	Wed 08:30	Wed 10:30	Wed 16:30	Thurs 08:30	Thurs 10:30	Thurs 14:30	Thurs 16:30	Fri 08:30	Fri 10:30	Fri 14:30
					X					

Marker's comments:

Problem	Marks	Obtained
1	16+19+17+8 = 60	
2	38	
3	22	
Total	120	

Extension certification:

This assignment has been given an extension and is now due on _____

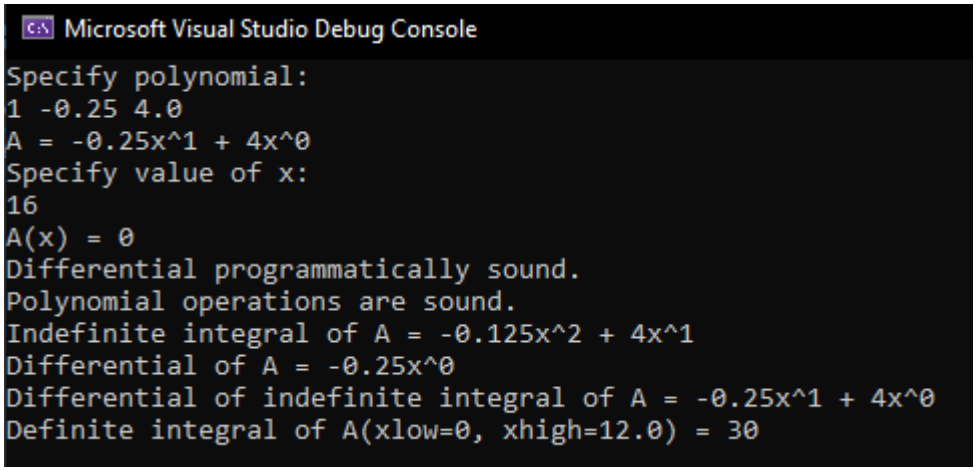
Signature of Convener: _____

Problem 1

```
#include "Polynomial.h"
#include <math.h>

double Polynomial::operator()(double aX) const {
	double lResult = 0;

	for (int i = fDegree; i >= 0; i--)
	{
		lResult += fCoeffs[i] * pow(aX, i);
	}

	return lResult;
}

Polynomial Polynomial::getDifferential() const {
	Polynomial lResult;

	for (int i = fDegree; i >= 0; i--)
	{
		lResult.fCoeffs[i - 1] = fCoeffs[i] * i;
	}

	if (fDegree - 1 >= 0)
	{
		lResult.fDegree = fDegree - 1 <= MAX_POLYNOMIAL ? fDegree - 1 : MAX_POLYNOMIAL;
	}
	else
	{
		lResult.fDegree = 0;
	}

	return lResult;
}

Polynomial Polynomial::getIndefiniteIntegral() const {
	Polynomial lResult;

	if (fDegree + 1 <= MAX_POLYNOMIAL)
	{
		lResult.fDegree = fDegree + 1;
	}
	else
	{
		lResult.fDegree = MAX_POLYNOMIAL;
	}

	for (int i = lResult.fDegree - 1; i >= 0; i--)
	{
		lResult.fCoeffs[i + 1] = fCoeffs[i] / ((double)i + 1);
	}

	return lResult;
}
```

```
double Polynomial::getDefiniteIntegral(double aXLow, double aXHigh) const {
	double lLow = 0;
	double lHigh = 0;

	for (int i = fDegree; i >= 0; i--)
	{
		lLow += fCoeffs[i] / ((double) i + 1) * pow(aXLow, i + 1);
	}

	for (int i = fDegree; i >= 0; i--)
	{
		lHigh += fCoeffs[i] / ((double)i + 1) * pow(aXHigh, i + 1);
	}

	return lHigh - lLow;
}
```

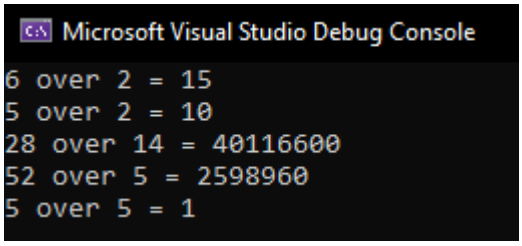
Output



Problem 2

```
#include "Combination.h"

Combination::Combination(unsigned int aN, unsigned int aK) {
	fN = aN;
	fK = aK;
}

unsigned int Combination::getN() const {
	return fN;
}

unsigned int Combination::getK() const {
	return fK;
}

unsigned long long Combination::operator()() const {
	if (fK > fN) return 0;
	if (fK == fN) return 1;

	unsigned long long lResult = 1;
	for (int k = 1; k <= fK; k++)
	{
		lResult *= ((unsigned long long)fN - k + 1);
		lResult /= k;
	}
	return lResult;
}
```

Output



Problem 3

```
#include "BernsteinBasePolynomial.h"
#include <math.h>

BernsteinBasePolynomial::BernsteinBasePolynomial() : fFactor(0, 0) {}

BernsteinBasePolynomial::BernsteinBasePolynomial(unsigned int aV, unsigned int aN) : fFactor(aN,
aV) {}

double BernsteinBasePolynomial::operator()(double aX) const {
	return fFactor() * pow(aX, fFactor.getK()) * pow((1 - aX), (fFactor.getN() -
fFactor.getK()));
}
```

Output